transaction that includes a plurality of modular elements individually protected by cryptographic security codes. --

In the drawings:

Please substitute the enclosed FIG. 2 for the corresponding figure filed with the application and please add the enclosed FIGS. 6-12, which correspond to figures in Appendix C filed with the application. Applicant has amended FIG. 2 to include the legend "FIG. 2" and the reference number "39." No new matter has been added.

REMARKS

The invention of claims 3 and 37 as amended provides an electronic commerce system and method in which a client computer transmits to a server computer over a public packet switched communications network an order acceptance request comprising a plurality of terms or conditions of a proposed offer for a purchase. The order acceptance request comprises a discrete message that includes a plurality of modular elements (one of which may be, for example, a coupon or gift certificate) individually protected by cryptographic security codes. The server computer processes the order acceptance request based on pre-programmed criteria, including authentication of the cryptographic security codes and examination of the modular elements of the discrete message individually protected by the cryptographic security codes. Based on the processing of the order acceptance request, the server computer transmits to the client computer an order acceptance response based on the pre-programmed criteria. The order acceptance response, which comprises a discrete message transmitted during a negotiation phase of a transaction, also includes a plurality of modular elements individually protected by cryptographic security codes.

As is discussed in the application as filed at page 9, lines 20-32, the individual protection of a modular element, such as a coupon or gift certificate, can allow the client computer and the server computer to efficiently store and forward the modular element together with its protection codes. For example, an order acceptance request may contain a digital coupon or gift certificate, individually protected by a protection code, that the client computer has obtained from a third party:

> The integrity of modular elements of order acceptance request 16 and order acceptance response 18 can be separately protected by protection codes embedded within the protected modular element. The protection codes can be implemented using digital signatures or message authentication codes or other well-known cryptographic security techniques. The embedding of these codes within modules of the messages enables client computer 12 and server 14 to efficiently store and forward the modular elements together with their protection codes. For example, an order acceptance request 16 may contain a digital coupon, protected by a protection code, that client computer 12 has obtained from a third party.

The Examiner contends that Sirbu, col. 12, lines 35-39, teaches the use of modular elements individually protected by cryptographic security codes, and provides the Kerberos tickets (discussed at column 12, lines 35-39) as an example of such individual protection. Sirbu, however, discusses the use of Kerberos tickets to protect individual phases of a transaction, rather than to protect individual modular elements of a discrete message (see col. 12, lines 35-39. emphasis added):

> Earlier, we showed how those signatures are used in the payment phase of the transaction. However, the transaction may also use Kerberos tickets. The transaction involves several phases, for price negotiation, goods delivery, and payment.

Nowhere does Sirbu suggest that Kerberos tickets may be used to protect a modular element of a discrete message, such as a coupon or a gift certificate that has been incorporated into a discrete message containing other components. Nor does Sirbu suggest that Kerberos tickets can allow a computer to efficiently store and forward a modular element together with its protection codes, such as a digital coupon or gift certificate that the computer has obtained from a third party.

Moreover, the claims as amended require the negotiation phase of a transaction to include transmission of an order acceptance response (in response to the order acceptance request) that includes modular elements individually protected by cryptographic security codes. Nowhere does Sirbu describe or suggest such an order acceptance response transmitted during the negotiation phase of a transaction (i.e., before acceptance of the terms or conditions by both computers) that includes modular elements individually protected by cryptographic security codes.

Because the Sirbu patent fails to describe or suggest the combination of a discrete order acceptance request message and a discrete order acceptance response message each of which includes individually protected modular elements, and instead discusses protection of a transaction involving several phases, claims 3 and 37 are both patentable over Sirbu.

Attached is a marked-up version of the changes being made by the current amendment.

Applicant asks that all claims be allowed. Enclosed is a Petition for Extension of Time with the required fee. Please apply any other charges or credits to Deposit Account No. 06-1050.

Respectfully submitted,

Date: June 25, 2001

James E. Mrose
James E. Mrose
Reg. No. 33,264

Fish & Richardson P.C.
601 Thirteenth Street, NW
Washington, DC 20005
Telephone: (202) 783-5070
Facsimile: (202) 783-2331

Version with markings to show changes made

In the specification:

Paragraphs added by the Response dated December 16, 1999 at page 30, line 4 before "What is claimed is:" have been amended as follows (double underlining represents newly added language):

The following description contains excerpts from a manual describing one implementation of the electronic commerce system outlined above.

Transact Architecture

Why Redesign the Architecture?

Transact 4 is conceived as the Transact release that:

- Provides a product with a complete set of features
- Exposes many internal layers to customization
- Allows customers to write applications

Transact 4 sets the standard that all similar Internet commerce products must meet because of this unique combination of inside-out exposure through the rich feature set and outside-in access through customization and the APIs. There are other commerce products that claim to provide similar ability to customize or extend features. The other products suffer in comparison to Transact 4 because they provide shell APIs with no built-in functionality. It is as though the programmers wrote an operating system API without implementing file systems, memory management, and other expected features of an operating system. Instead, the product designers expect the software owner to provide these features. Transact 4 achieves the difficult task of providing a fully functional Internet commerce operating system with all the hooks necessary to extend or replace core components.

Many owners will be satisfied using Transact as it is after installation, without any further customization than editing the screens with their favorite HTML editing tool. Sellers and their employees, especially, will appreciate the Commerce Center that gives them access to store registration and management. Access to core business logic in the product through APIs allows customers to write modules to extend product functionality without requiring recompilation, relinkage, or reinstallation of the product.

Examples of customization in Transact 4 are:

- Sellers and administrators can customize large portions of the user interface.
- Sellers and administrators can localize the product to the language of their choice, including those with multibyte character encoding.
- Sellers have a wide variety of payment options they can choose from in Transact.
- Core objects in the system can be tagged with extra or custom fields defined by the seller. The fields are retained in the system and are retrievable via the user interface and the APIs.

Examples of APIs available for customer applications are:

- The Buyer Profile API, designed for customers who wish to load the transaction database with existing customer information that they already have.
- The Fulfillment API, which stores the purchase history and provides the mechanism necessary for writing custom fulfillment applications.
- The Microtransactions API, which lets customers write their own prepay or postpay microtransaction applications with their own user interface. An owner can replace or extend any aspect of microtransaction functionality without affecting the core product.
- The Order Entry API, which lets customers implement their own catalog or shopping cart applications that deposit orders into Transact, bypassing the Transact buyer user interface.

Examples of extensible and replaceable modules in Transact 4 are:

- Payment modules - These modules are for enabling both payment brands, such as Visa, and payment agents, such as First USA Paymentech.
- Order Entry modules - These modules plug in to the order capture subsystem of Transact 4 that performs activities like inventory checks or payment card fraud checks.
- E-mail modules - Owners can replace the way Transact 4 sends e-mail to users, including the encryption method.
- Fax modules - Owners can replace the way Transact 4 sends faxes.
- Smart Pages modules - Owners who want to customize screens for each buyer can write a module to do this.
- Tax modules - Owners can replace or extend the way Transact computes sales tax.

The Payment API is a public API. All the other APIs are leveraged with the help of Open Market consulting services.

Foundations

Transact 4 is a collection of objects, components that contain those objects, and applications that use the component and object APIs. There are no monolithic applications in

Transact 4, instead, all Transact applications, such as all the buyer CGI scripts, are clients of component APIs. It is therefore easy to discuss scenarios where components may be replaced or extended without requiring changes to the basic applications.

Component Architecture

The component architecture of Transact 4 leverages the ideas and technology behind the Microsoft component object model (COM). COM describes a world where components define classes and classes implement interfaces. Interfaces are the common *lingua franca*; components and classes may be replaced, extended or revised without requiring client changes as long as they continue to implement the same set of interfaces.

The strength of COM is that it provides a mechanism for independently written software components to find each other through the registry, launch each other through dynamic linking, and a common language through their interfaces.

For example, someone wishing to write a new kind of payment instrument in Transact 4 does so by implementing a new component, assigning it a new class identifier, and ensuring that the component implements the appropriate payment authorization and settlement interfaces. The component is physically a shared library that can be dynamically linked into Transact applications. Transact 4 provides a registry where information is stored about the new component, such as its class ID, where it lives, and which interfaces it support. The next time a Transact application that needs available payment options is launched, it interrogates the registry to find the new component.

Transact 4 ships with some components, for example, e-mail and fax. Information about these components can be found in the registry as well. They are implemented as dynamically linked libraries with class identifiers and other necessary COM information. The effort required to replace or extend them is similar to the effort required to develop a new payment component.

Architectural Overview

Transact is an order capture and processing system that interacts with one or more payment modules that handle the payment for orders. In addition to handling orders, it provides features for sellers to manage their stores, an interface for buyers to complete and monitor purchases, and tools for an administrator to monitor Transact. All of the Transact subsystems operate with a common set of business objects:

Order object. This object represents a contract between the buyer and seller. The object captures all information relevant to the order, including the items being ordered, the terms and conditions, the payment information, and the fulfillment information. The moment of capture

occurs when the buyer clicks a "buy" button; until that moment the order information is stored in a shopping cart.

Invoice object. This object represents a fulfillment event, where goods or services are delivered, fully or partially, to the buyer. Any subset of goods or services in an order can be shipped, canceled, returned, or charged back. One invoice object represents one action to one or more items in an order object.

Principal object. A principal is an entity that can be authenticated. This object identifies the software component or person contacting Transact, so Transact can ascertain who the entity is and what it is allowed to do.

Payment instruction. The pmt Instruction object captures information concerning the mechanism for funds transfer between buyer and seller, such as a payment brand, authorization, or a payment agent.

The user interface and application logic, illustrated in FIG. 6, are interfaces to users. There is a library of common user interface screen components, for example, the order form or the name address block, that are shared among subsystems.

There are three groups of applications:

Buyer applications. This group implements all buyer activities, including order capture, order status, payment account establishment, payment account updates, Smart Statements, and so on.

Seller applications. This group implements all online business activities, such as managing the store, presenting statements, processing orders, and generating reports.

Administrator tools. This group implements Transact administrator activities, such as configuration, key management, and maintenance.

The user interfaces for these applications depends on a common set of libraries:

Smart Pages. This library provides a common set of HTML screen components, such as a name address block for rendering addresses. These are the screen components that provide a template mechanism for rendering HTML screens.

HTML/HTTP common services. These libraries provide a common set of server-side HTTP protocol and HTML services for handling requests, parsing query strings and form POST data, handling URLs, and so on.

The core subsystems, illustrated in FIG. 7, provide the base functionality on which the buyer, seller, and administrator applications are built. The core subsystems implement the functionality behind the Transact APIs.

They include:

Order-invoice subsystem. This subsystem provides services that accept and validate new orders, as well as update and report on the current state of orders and invoices in the system.

Store subsystem. This subsystem maintains the status of the business and provides a set of seller services related to order processing, including tax and shipping calculations.

Directory subsystem. This subsystem maintains a database of all Transact users, known as principals, and of all addresses when principals have addresses. For example, a buyer is a principal with associated addresses. A fulfillment module is a "user" of the system, but it has no address. This subsystem also provides a set of services for adding, deleting, retrieving, and authenticating users.

Payment subsystem. This subsystem provides payment services, such as authorization and settlement. The payment subsystem hides most of the details of particular payment instruments, so new instruments can be added here with minimal or no changes to other subsystems.

Logging subsystem. This subsystem provides a central event logging facility for all other subsystems.

All subsystems depend on a set of common utility libraries for string handling, message catalogs, hash functions, and so on. All subsystems gather configuration information from a common facility called registry.asc.

Core Business Objects

This section details the core objects that are sued by nearly all subsystems.

The business objects mentioned at the beginning of this chapter are just portable data structures that represent the contents of business documents such as orders, invoices, and receipts. They are not backed by a database, though they are certainly stored in databases, and can be carried anywhere a byte stream can be handled, for example, MIME attachments in e-mail, HTTP, flat files, and so on. Examples of these objects are the order, orderItem, invoice, invoiceItem, and address objects.

Transact subsystems provide service objects. Service objects accept, process, and find business objects such as orders, invoices, payments, and addresses. Each of the business objects implements the following methods:

Mutators. These methods are used to change or set object state. The methods may perform basic error checking such as string length checking, or they may prevent updates for objects in a read-only state.

Accessors. These methods read or get object states.

Serialization and deserialization. These methods write and read the object contents to a byte stream. The stream is portable; objects serialized on one platform are readable on all other platforms. The stream is reasonably compact, as the objects have to be passed over slow dial-up links, for example, for the order capture subsystem. The stream may contain various consistency checks, such as a unique type identifier and a checksum. The stream representation has a version number, so that objects can be revised, yet be backwards compatible to old streams.

The serialization and deserialization methods are inspired by the Microsoft structured storage for persistent COM objects, though our approach is restricted to simple byte streams.

The service and business objects do not have any user interface components. The presentation of the data contained within the objects is handled by Smart Pages.

All strings in all objects support the Unicode standard for encoding characters. For places where strings show up in external interfaces, the UTF-8 variable length encoding, which is backward compatible with 7-bit ASCII, is preferred.

Order and OrderItem Objects

An order object, illustrated in FIG. 8, represents the contract between buyer and seller for the items ordered and the terms and conditions of the sale. An orderItem object represents a line item in an order. The order object contains zero or more orderItem objects.

Open Market made some simplifying design assumptions for the order object:

- Each order has a single billing address and a single shipping address. If you want to ship different items to different addresses, create multiple orders per destination address.
- Each order is for items from a single store or business. If a buyer wants to buy items from multiple stores, then he or she creates multiple orders.

Note: The above assumptions about the objects do not prevent us from implementing a multiple-address, multiple-store user interface for buyers.

Current order status is maintained by the order-invoice subsystem. That subsystem provides services for looking up existing orders, adding new orders, and changing order status.

Invoice and InvoiceItem Objects

An invoice object, illustrated in FIG. 9, is a record of fulfillment that is created every time a shipment or cancellation occurs.

An invoiceItem is one of the items in the invoice that was shipped or canceled. We make the simplifying assumption that every invoice refers to only one order. An order can have multiple invoices associated with it, one for each fulfillment activity related to the order. An invoice has a single billing and shipping address. An invoice also has a single pmtInstruction object associated with it.

Payment Instruction Object

A pmtInstruction object, illustrated in FIG. 10, is created when the user submits an order form. It encapsulates the payment contact between the buyer and seller, the payment obligation of the buyer, the payment instrument that the buyer wants to use and the seller agrees to accept, as well as the state of funds transfer between the buyer and seller.

A Transact payment agent module uses methods to accomplish a payment authorization and funds transfer. The payment object carries the payment brand identifier of the payment module.

Address Object

The address object, illustrated in FIG. 11, is used to hold postal address, e-mail address, and other contact information. The address object is contained within other objects, such as the order and invoice objects.

Applications

This section provides further information about the Transact application software. It is divided according to user class, however, many applications are used by a combination of user classes with different information available to different classes.

Buyer Applications

The buyer applications group together user interface, API, and functional logic for buyer tasks into CGI scripts. Four of the major applications are described below. Other buyer applications include:

- payinfo.cgi - allows buyers to register payment cards
- subsummary.cgi - view of subscription history
- microbuy.cgi - features for making microtransactions
- microbal.cgi - view of current prepay balance for microtransactions

Order Form Application

The order-form.cgi script provides the user interface and application logic for composing and capturing orders from the buyer. The output of the order capture process is a complete order, which is handed off to the order-invoice subsystem for processing. This module provides an API that supports the implementation of ActiveX and Java user interfaces for order capture. It can also support the implementation of remote user interfaces, where the interface runs on the content server, but uses order capture services provided by Transact.

This module implements an HTML user interface on top of the Order Entry API. This user interface implements an HTML version of everything needed to capture an order: processing incoming digital offers (DOs), maintaining shopping carts, and collecting payment

information. This user interface is commonly referred to as the buyer flow, and represents most of the functionality in the payment.cgi script that was in Transact 3.0.

Modularizing order capture generates implementation flexibility. In the future, it allows for specialized order capture modules for particular applications. These variants can run simultaneously on a single Transact server, each sharing the common core for order processing and service. Example modules include a:

- Specialized retail module
- Business-to-business module
- Buyer work flow order approval module
- Support for negotiated prices module

Statements and Receipts Application

The smart-statement.cgi and smart-receipt.cgi applications provides a user interface and application logic for servicing the buyer's order after it has been captured. Order service tasks include:

- Displaying the Smart Statement
- Processing order and invoice status
- Handling payment failures
- Making returns and credits

User Information Module

The user information module implements a user interface for buyers to perform registration maintenance tasks through userinfo.cgi, including:

- Registering for the first time
- Changing registration information

Seller Applications

Seller applications implement the user interface and application logic for sellers to register and manage their stores. The Commerce Center is run by index.cgi. Sellers use smart-receipt.cgi and smart-statement.cgi to monitor and fulfill orders. The reconcile.cgi script is used by both buyers and merchants to review and solve payment card problems. There are four main groups of seller applications:

Store Registration Module

The applications store-registration.cgi, oaa_select_tax.cgi, oaa_vat_register, and transport.cgi let sellers create stores and specify various store parameters such as:

- Store locale
- Store currency

- Acceptance of purchase orders or microtransactions
- Store payment agent parameters
- Shipping options
- Tax options

The transport.cgi script lets a seller designate a fulfillment party to perform shipping activities. This module allows for partial shipment and cancellations and makes extensive use of the Transact Fulfillment API.

Reporting Module

This module enables a variety of seller reports, including the ability to define new reports. Examples of reports are:

- Customer listings
- Orders placed at your store
- Payment details and processor information for reconciliation purposes
- Tax reporting on the amount of tax charged on an order or invoice.

Tax audit reports are still downloaded directly from the transaction database.

The supporting architecture for the reporting module is to extract a reporting database with a subset of information from the main database in the following way.

Transact run jobs that put "slices" of the seller's data in Microsoft access format onto the Transact file system in a specific directory. These extracted files are encrypted because payment card numbers might be in the data stored.

The remote client browses this directory to see which new files are present. The client then makes a secure request for the files with the Transact reporting client. The reporting module authenticates the seller, validates the request, decrypts the files, and opens the files to download the data.

Sellers retrieve the Access database and store it on their desktop. Reports can be generated on the data using any reporting tool that supports Access.

Microtransactions Module

The microtransactions subsystem separates fulfillment and settlement by allowing authenticated buyers to either prepay or postpay for access to digital goods.

It introduces the tab object, which keeps a tally of a buyer's charges and credits at one store. The microtransaction configuration object records settings for microtransactions for one store. The microtransactionItem object is analogous to an orderItem object, representing one purchase of one digital good.

In both the prepay and postpay case, the buyer chooses a digital good marked as a microtransaction. The microbuy.cgi script processes the offer by debiting the buyer's tab, then redirecting the buyer to the digital receipt, creating an effect of going directly to the information. For postpay accounts, there is an additional step, performed through the Microtransactions API, that aggregates the microtransaction purchases into an order. The order is then passed to the order-invoice subsystem for processing.

When a buyer establishes a prepay tab, the amount is authorized and settled before delivery of the goods occurs. When a buyer is approved off line for a postpay account, the buyer delivery occurs before payment is completed.

Taxes are calculated on the entire prepay amount before it is authorized, so taxes are included in the authorized amount. In the postpay case, taxes are applied to each digital good, then the total for that item is added to the tab.

A second script, microbal.cgi, allows buyers and merchants to review the buyer's tab.

Subscription Module

Sellers and buyers also use the scripts concerned with subscriptions, which allow for purchasing and accessing online content over an unlimited or a specified length of time. The subscriptions component takes the order from the order capture subsystem before the moment of capture. It analyzes the digital offers an enters data into the periodical, subscription, installment, and installment_plan database tables. Invoices, created by an ongoing job run by Transact, trigger payment at the appropriate intervals.

Subscription applications include:

- authorize.cgi - This application, used for many reasons within Transact, allows users access to Transact scripts and content. A buyer only needs to log in or register themselves once before viewing content.
- subscription.cgi - The subscription application verifies that the buyer has paid for a subscription. It then authorizes them to access content areas they have paid for.
- subsummary.cgi - Buyers can review subscriptions they have purchased.
- sub-manage.cgi - Sellers can manage prices or cancellations of subscriptions.
- sub-acl.cgi - Sellers use this application to mange access control lists (ACL) to subscriptions.

Administrator Applications

Administrator tools provide the user interface, external interfaces, and application logic for Transact administrators to run their system. There are three modules.

Installation Module

This module is structured so that all queries of the installer are made before the actual install. The install module decompresses all the servers through untar and places them appropriately on the host machines.

Upgrade Module

After an installation is complete, the upgrade module can be invoked to copy all relevant data from an existing Transact 3.0 system to the new installation. This data includes:

- The transaction database
- Payment processor information
- Relevant Transact and satellite server configuration information
- Web server configuration information, including the migration from Open Market Secure WebServer to Netscape Enterprise Server

Administration Module

The third module is the forms-based administration screens and functionality. This module has not been extensively upgraded for Transact 4. Relevant applications include:

- index.cgi - Presents the main FBA screen.
- configure-agents.cgi - Controls configuration of payment agents.
- acct-validate.cgi - This critical installation step is used to configure the payment processor that is used for address verifications (AVS).
- payment-errors.cgi - Lists errors in payment card transactions.
- logmessage.cgi - Displays log messages in HTML format.
- info-deliver.cgi - Presents the administrator tools menu.

Application Program Interfaces (APIs)

The objects, applications, and subsystems are connected through the use of APIs as shown in FIG. 12. Note that some of the APIs in the scheme below are internal: internal Payment, user service, and store service. Subscription authentication, settlement, and fax advice of orders are conducted with the help of HTTP. Descriptions of the APIs from a designer's perspective can be found in "APIs" on page 69.

Buyer Profile API

The Buyer Profile API, formerly called the customer database API, which enables bulk loading of buyer information from an external database to the Transact database. This API is not enhanced in Transact 4.

Fulfillment API

The Fulfillment API has two major purposes:

- Order and invoice query

° Invoice creation and entry into the database

Order and invoice query are used by applications that wish to query the status of orders that match specific criteria such as:

° Order number

° Range of order numbers

° Date range

° Buyer name or address

Invoices for retrieved orders can be queried as well.

Transact buyer and seller applications make extensive use of the order and invoice query mechanisms in the Fulfillment API.

Invoice creation and entry are used by applications that wish to augment the shipping module, either by presenting bulk shipping functionality, or by custom shipping user interfaces. An invoice is created for every shipping or cancellation activity, partial or full. An associated payment instruction object is created and funds transfer initiated, if Transact is expected to do settlement. The invoice is then entered into the transaction database.

Order Entry API

This API is the heart of the order capture subsystem; it is an API that separates the order capture function from the user interface. It also has the important property that it can work securely with nonsecure clients, that is, it doesn't depend on any client to do the final calculations.

The API uses an offer-counter-offer style. Clients create order objects representing proposed orders, then the subsystem gets the order proposals, verifies them, and creates another order object representing the counter-offer, which also includes calculated values for tax, shipping, and discounts. The counter-offer order may include a set of restrictions from which the buyer has to select, such as a limited number of shippers, before the seller accepts the final order.

The order entry API invokes the underlying services of stores, authentication, and payment as necessary to verify the order contents, calculate various derived values, and accept a valid order for injection. Once a valid order is captured, it is handed off to the order injector for processing by the rest of the system.

Payment API

The Payment API has two parts, one that works internally with Transact, and one that allows system integrators or Open Market consulting to add new payment modules.

Transact Internal Payment API

There is a large and complex part of payment internal to Transact. The internal Payment API is a collection of APIs to encapsulate those portions of Transact that know about the details of payment.

This internal API makes it possible to create a payment instruction object and move it through the states of authorization and funds transfer. The information provides includes:

- Information about the buyer payment instrument
- Brand issuing the instrument
- Authentication method for the payment instrument
- Payment agent and acquirer
- Amount and currency of funds transfer

External Payment API

Open Market offers only a fraction of all the payment options possible with Transact. The external Payment API provides a set of modules that allow new forms of payment to be added to Transact without changes to the rest of the Transact installation. Transact payment modules may implement one or both of the following objects:

Payment brand. An entity that issues payment instruments and backs them, for example, Visa. A module that implements a particular payment brand may elect to use an existing payment agent for authorization and settlement. For example, a new Winnie-the-Pooh brand credit card may elect to use a First USA Paymentech payment agent, because First USA can support the new card.

Payment agent. An entity that can manage funds transfer between the buyer payment instrument and the seller, e.g., payment processors, a SET payment gateway, a Mondex value acquirer, or a billing system.

The module implements the authorization and settlement methods. Replaceable payment agents allow for a variety of payment mechanisms, each of which supports a radically different paradigm for authorization and settlement.

Security on the Internet is a concern for many buyers and sellers. As a Transact owner, it is part of your role to reassure sellers and buyers about the relative safety of the Internet and the specific ways in which Transact protects them. The power of the World Wide Web and the Internet on which it rests does depend on its common protocol and open architecture, but that doesn't mean that Transact cannot protect the transmissions and data necessary to conduct business. This chapter looks at areas where security is a concern and the way Transact responds. Open Market has put a great deal of effort into successfully securing Web transactions, so that consumers and businesses can feel confident that abuses are averted.

It's also important to encourage businesses and consumers worried about security issues on the Internet to make a realistic assessment of the risks. For example, most visitors to a shopping mall don't think twice about giving their payment card to a waiter or sales clerk, even though the number is being registered in a system that could be unsafe, or that a copy of the charge receipt could end up blowing out of the dumpster behind the building. There is a misperception that it is easier to steal profitable information on the Internet than it is during physical business transactions, which is not true.--

In the claims:

Claims 3 and 37 have been amended as follows:

3. (Amended) An electronic commerce system, comprising:

a client computer; and

a server computer;

the client computer and the server computer being interconnected by a public packet switched communications network;

the client computer being programmed to transmit to the server computer an order acceptance request comprising a plurality of terms or conditions of a proposed offer for a purchase, the order acceptance request comprising a discrete message that includes a plurality of modular elements individually protected by cryptographic security codes;

the server computer being programmed to process the order acceptance request based on pre-programmed criteria, including authentication of the cryptographic security codes and examination of the modular elements of the discrete message individually protected by the cryptographic security codes, and, based on the processing of the order acceptance request, to transmit to the client computer an order acceptance response based on the pre-programmed criteria, the order acceptance response comprising a discrete message transmitted during a negotiation phase of a transaction that includes a plurality of modular elements individually protected by cryptographic security codes.

37. (Amended) A method of processing order acceptance requests in an electronic commerce system, comprising a client computer and a server computer interconnected by a public packet switched communications network, the method comprising;

receiving at the server computer an order acceptance request transmitted by the client computer comprising a plurality of terms or conditions of a proposed offer for a purchase, the order acceptance request comprising a discrete message that includes a plurality of modular elements individually protected by cryptographic security codes;

processing the order acceptance request based on pre-programmed criteria, including authentication of the cryptographic security codes and examination of the modular elements of the discrete message individually protected by the cryptographic security codes; and

based on the processing of the order acceptance request, transmitting to the client computer an order acceptance response based on the pre-programmed criteria, the order acceptance response comprising a discrete message transmitted during a negotiation phase of a transaction that includes a plurality of modular elements individually protected by cryptographic security codes. --